

## Exp 4

**AIM:** Hands on Solidity Programming Assignments for creating Smart Contracts.

### Theory:

#### 1. Primitive Data Types, Variables, Functions – pure, view

In Solidity, primitive data types form the foundation of smart contract development. Commonly used types include:

- **uint / int**: unsigned and signed integers of different sizes (e.g., uint256, int128).
- **bool**: represents logical values (true or false).
- **address**: holds a 20-byte Ethereum account address, often used for storing user accounts or contract addresses.
- **bytes / string**: store binary data or textual data.

Variables in Solidity can be **state variables** (stored on the blockchain permanently), **local variables** (temporary, created during function execution), or **global variables** (special predefined variables such as msg.sender, msg.value, and block.timestamp).

Functions allow execution of contract logic. Special types of functions include:

- **pure**: cannot read or modify blockchain state; they work only with inputs and internal computations.
- **view**: can read state variables but cannot alter them. This classification helps optimize gas usage and enforces function integrity.

#### 2. Inputs and Outputs to Functions

Functions in Solidity can accept input arguments and return one or more output values. Inputs enable users or other contracts to pass data into the contract, while outputs make it possible to return results after computation. For example, a function can accept an amount in Ether and return whether the transfer was successful. Solidity also allows named return variables, which improve readability and debugging.

#### 3. Visibility, Modifiers and Constructors

- **Function Visibility** defines who can access a function:

- **public**: available both inside and outside the contract.
- **private**: only accessible within the same contract.
- **internal**: accessible within the contract and its child contracts.
- **external**: can be called only by external accounts or other contracts.

- **Modifiers** are reusable code blocks that change the behavior of functions. They are often used for access control, such as restricting sensitive functions to the contract owner (`onlyOwner`).
- **Constructors** are special functions executed only once during contract deployment. They initialize important values, such as setting the deploying account as the owner of the contract.

### 3. Control Flow: if-else, loops

Control flow in Solidity is similar to traditional programming languages:

- **if-else** allows conditional decision-making in contract logic, e.g., checking if a balance is sufficient before transferring funds.
- **Loops** (for, while, do-while) enable repeated execution of code. For example, iterating through an array of users. However, loops must be used carefully, as excessive iterations increase gas consumption, potentially making the contract expensive to execute.

### 5. Data Structures: Arrays, Mappings, Structs, Enums

- **Arrays**: Can be fixed or dynamic and are used to store ordered lists of elements. Example: an array of addresses for registered users.
- **Mappings**: Key-value pairs that allow quick lookups. Example: `mapping(address => uint)` for storing balances. Unlike arrays, mappings do not support iteration.
- **Structs**: Allow grouping of related properties into a single data type, such as creating a struct `Player {string name; uint score;}`.
- **Enums**: Used to define a set of predefined constants, making code more readable. Example: `enum Status { Pending, Active, Closed }`.

### 6. Data Locations

Solidity uses three primary data locations for storing variables:

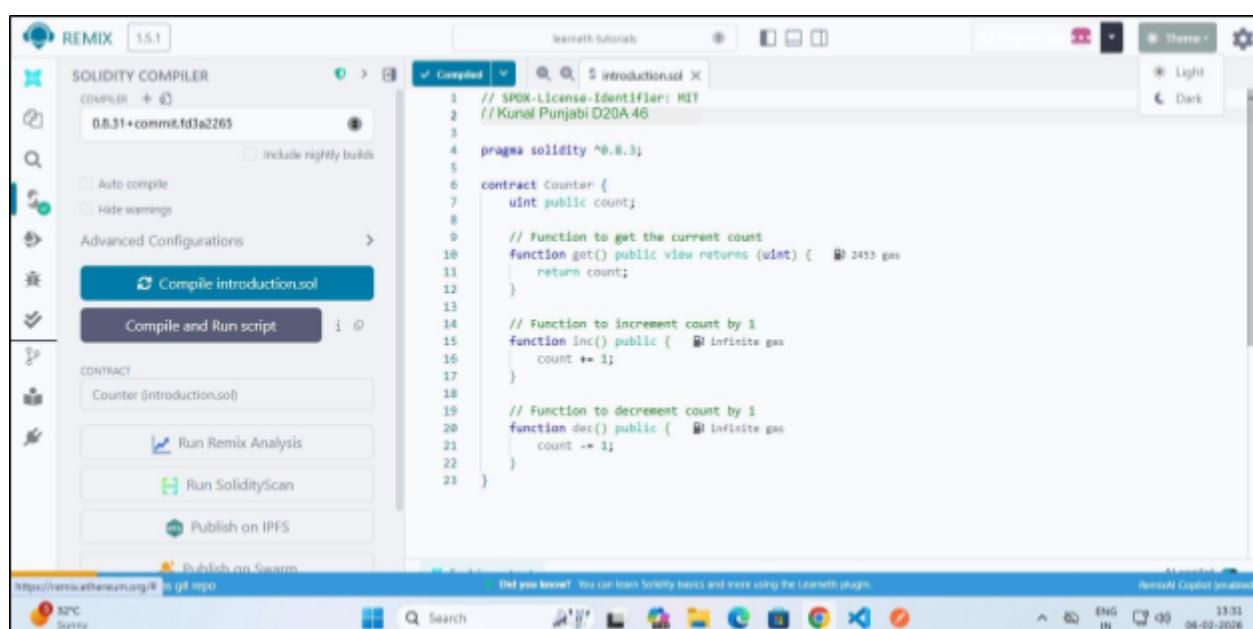
- **storage**: Data stored permanently on the blockchain. Examples: state variables.
- **memory**: Temporary data storage that exists only while a function is executing. Used for local variables and function inputs.
- **calldata**: A non-modifiable and non-persistent location used for external function parameters. It is gas-efficient compared to memory. Understanding data locations is essential, as they directly impact gas costs and performance.

## 7. Transactions: Ether and Wei, Gas and Gas Price, Sending Transactions

- Ether and Wei:** Ether is the main currency in Ethereum. All values are measured in Wei, the smallest unit (1 Ether =  $10^{18}$  Wei). This ensures high precision in financial transactions.
- Gas and Gas Price:** Every transaction consumes gas, which represents computational effort. The gas price determines how much Ether is paid per unit of gas. A higher gas price incentivizes miners to prioritize the transaction.
- Sending Transactions:** Transactions are used for transferring Ether or interacting with contracts. Functions like `transfer()` and `send()` are commonly used, while `call()` provides more flexibility. Each transaction requires gas, making efficiency in contract design very important.

### Implementation:

#### Tutorial - 1 (compile)



The screenshot shows the Remix IDE interface. On the left, the Solidity Compiler section displays the version 0.8.3+commit.fd3a2265 and includes options for Auto compile, Hide warnings, and Advanced Configurations. A button labeled "Compile introduction.sol" is highlighted. Below it, the CONTRACT section shows "Counter (Introduction.sol)". There are buttons for "Run Remix Analysis", "Run SolidityScan", and "Publish on IPFS". At the bottom, there's a "Publish on Swarm" button. The central area contains the Solidity code for the Counter contract:

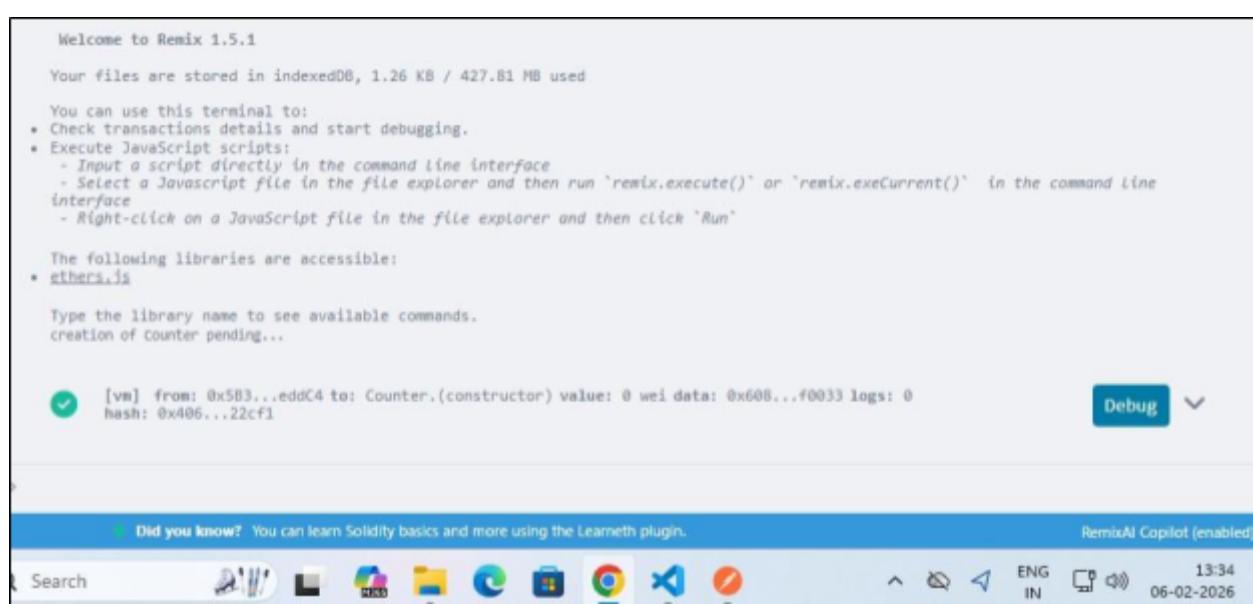
```
// SPDX-License-Identifier: MIT
// Kunal Panjabi D20A 46
pragma solidity ^0.8.3;

contract Counter {
    uint public count;

    // Function to get the current count
    function get() public view returns (uint) {
        return count;
    }

    // Function to increment count by 1
    function inc() public {
        count += 1;
    }

    // Function to decrement count by 1
    function dec() public {
        count -= 1;
    }
}
```



Welcome to Remix 1.5.1

Your files are stored in indexedDB, 1.26 KB / 427.81 MB used

You can use this terminal to:

- Check transactions details and start debugging.
- Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run `'remix.execute()'` or `'remix.exeCurrent()'` in the command line interface
  - Right-click on a JavaScript file in the file explorer and then click 'Run'

The following libraries are accessible:

- `ethers.js`

Type the library name to see available commands.  
creation of Counter pending...

[vm] from: 0x5B3...eddC4 to: Counter.(constructor) value: 0 wei data: 0x600...f0033 logs: 0 hash: 0x406...22cf1

Debug

Did you know? You can learn Solidity basics and more using the Learneth plugin.

RemixAI Capilot (enabled)

## Tutorial - 1 (get)

Balance: 0 ETH

dec

inc

count

get

---

Low level interactions

CALldata

Transact

### Tutorial - 1 (inc)

COUNTER AT 0xF8E...9FBE8 (MEMORY)

Balance: 0 ETH

dec  
inc  
count  
get

0: uint256: 1

Low level interactions i

CALldata

Transact

### Tutorial - 1 (dec)

COUNTER AT 0xF8E...9FBE8 (MEMORY)

Balance: 0 ETH

dec  
inc  
count  
get

0: uint256: 0

Low level interactions i

CALldata

Transact

## Tutorial - 2

The screenshot shows the REMIX IDE interface. On the left, the 'LEARNETH' sidebar displays a 'Tutorials list' with '2. Basic Syntax' selected. The main workspace shows a Solidity code editor with the following content:

```
// SPDX-License-Identifier: MIT
// compiler version must be greater than or equal to 0.8.3 and less than 0.9.0
pragma solidity ^0.8.3;
//Pratik Pandey D20A 37
// Kunal Punjabi D20A 46
contract MyContract {
    string public name = "Alice";
}
```

The status bar at the bottom indicates 'Well done! No errors.'

## Tutorial - 3

The screenshot shows the REMIX IDE interface. On the left, the 'LEARNETH' sidebar displays a 'Tutorials list' with '3. Variables' selected. The main workspace shows a Solidity code editor with the following content:

```
address public addr = 0xC35b7d91545BEF540aDe6068dFe2f44EBfa733c;
address public newaddr = 0x0000000000000000000000000000000000000000;
int public neg = -37;
uint public newU = 8;

// Default values
// Unassigned variables have a default value
bool public defaultBool; // false
```

The status bar at the bottom indicates 'Well done! No errors.'

## Tutorial - 4

A list of all Global Variables is available in the [Solidity documentation](#).  
Watch video tutorials on [State Variables](#), [Local Variables](#), and [Global Variables](#).

**★ Assignment**

1. Create a new public state variable called `blockNumber`.
2. Inside the function `increment()`, assign the value of the current block number to the state variable `blockNumber`.

Tip: Look into the global variables section of the Solidity documentation to find out how to read the current block number.

Check Answer Show answer Next Well done! No errors.

```
10  uint public blocknumber;
11
12
13
14
15  uint i = 456;
16  blockNumber = block.number;
17  // Here are some global variables
18  uint timestamp = block.timestamp; // Current block timestamp
19  address sender = msg.sender; // Address of the caller
20 }
```

Explain contract Listen on all transactions Filter with transaction hash or ad... AI copilot

Note: The called Function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 5

**★ Assignment**

1. Create a public state variable called `b` that is of type `bool` and initialize it to `true`.
2. Create a public function called `get_b` that returns the value of `b`.

Check Answer Show answer Next Well done! No errors.

```
18  bool public b = true;
19
20  function get_b() public view returns (bool) { 2539 gas
21      return b;
22  }
23 }
```

Explain contract Listen on all transactions Filter with transaction hash or ad... AI copilot

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 6

**★ Assignment**

Create a function called `addtox` that takes the parameter `y` and updates the state variable `x` with the sum of the parameter and the state variable `x`.

Check Answer Show answer Next Well done! No errors.

```
17  function addToX2(uint y) public { infinite gas
18      x = x + y;
19  }
20 }
```

Explain contract Listen on all transactions Filter with transaction hash or ad... AI copilot

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 7

Watch a video tutorial on Function Modifiers.

**Assignment**

- Create a new function, `increase`, in the contract. The function should take an input parameter of type `uint` and increase the value of the variable `x` by the value of the input parameter.
- Make sure that `x` can only be increased.
- The body of the function `increase` should be empty.

Tip: Use modifiers.

Check Answer Show answer Next Well done! No errors.

```
55 modifier greatthan(uint y){  
56     require(y > 0, "Input parameter not greater than 0");  
57     _;  
58 }  
59 modifier actualIncrease(uint y){  
60     _;  
61     x = x + y;  
62 }  
63 // Kunai Punjabi D20A-46  
64 function increase(uint y) public onlyOwner greatthan(y) actualIncrease(y){  
65     _;  
66 }
```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad... AI copilot

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 8

**Assignment**

Create a new function called `returnTwo` that returns the values `-2` and `true` without using a return statement.

Check Answer Show answer Next Well done! No errors.

```
80 function returnTwo() public pure returns(int p, bool q) {  
81     p = -2;  
82     q = true;  
83 }
```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad... AI copilot

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 9

Create a new function in the `base` contract called `testInternalVar` that returns the values of all state variables from the `base` contract that are possible to return.

Check Answer Show answer Next Well done! No errors.

```
67 function testInternalVar() public view returns (string memory a, string memory b){  
68     return (internalVar, publicVar);  
69 }
```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad... AI copilot

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 10

**Assignment**

Create a new function called `evenCheck` in the `ifelse` contract:

- That takes in a `uint` as an argument.
- The function returns `true` if the argument is even, and `false` if the argument is odd.
- Use a ternary operator to return the result of the `evenCheck` function.

Tip: The modulo (%) operator produces the remainder of an integer division.

Check Answer Show answer Next Well done! No errors.

```
18     // }  
19     // return 2;  
20  
21     // shorthand way to write if / else statement  
22     return _x < 10 ? 1 : 2;  
23 }  
24 // Kunai Punjabi D20A-46  
25 function evenCheck(uint a) public pure returns (bool){  
26     return a % 2 == 0 ? true : false;  
27 }
```

Explain contract

0 Listen on all transactions Filter with transaction hash or ad... AI copilot

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 11

The screenshot shows a Solidity tutorial interface. On the left, there's a sidebar with 'Tutorials list' and 'Syllabus'. The main area displays a tutorial titled '7.2 Control Flow - Loops' (11 / 19). It includes a section on 'break' loops, a video link, and an 'Assignment' section with three tasks related to loops. Below the assignment is a button to 'Check Answer' or 'Show answer'. On the right, the Solidity code for a 'Loop' contract is shown:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract Loop {
5     // Kunal Panjabi D20A 46
6     uint public count;
7
8     function loop() public {    // infinite gas
9         // for loop
10        for (uint i = 0; i < 10; i++) {
11            if (i == 5) {
12                // Skip to next iteration with continue
13                continue;
14            }
15            if (i == 5) {
16                // Exit loop with break
17                break;
18            }
19            count++;
20        }
21    } // while loop
22 }
```

Below the code, there's an 'Explain contract' button and an 'AI copilot' toggle. A note at the bottom says: 'note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.'

## Tutorial - 12

The screenshot shows a Solidity tutorial interface. On the left, there's a sidebar with 'Tutorials list' and 'Syllabus'. The main area displays a tutorial titled '8.1 Data Structures - Arrays' (12 / 19). It includes a note about moving array elements, a section on 'Array length', and a video link. Below is an 'Assignment' section with two tasks related to arrays. A 'Check Answer' or 'Show answer' button is present. On the right, the Solidity code for an 'Arrays' contract is shown:

```
7 uint[] public arr2 = [1, 2, 3];
8 // Kunal Panjabi D20A 46 D20A 37
9 uint[3] public arr3 = [0, 1, 2];
10 // Fixed sized array, all elements initialize to 0
11 uint[10] public myFixedSizeArr;
12
13 function get(uint i) public view returns (uint) {    // infinite gas
14     return arr[i];
15 }
16
17 // Solidity can return the entire array.
18 // But this function should be avoided for
19 // arrays that can grow indefinitely in length.
20 function getArr() public view returns (uint[3] memory) {    // infinite gas
21     return arr3;
22 }
23
24 function push(uint i) public {    // 46820 gas
25     // Append to array
26     // This will increase the array length by 1.
27     arr.push(i);
28 }
```

Below the code, there's an 'Explain contract' button and an 'AI copilot' toggle. A note at the bottom says: 'note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.'

## Tutorial - 13

We can use the delete operator to delete a value associated with a key, which will set it to the default value of 0. As we have seen in the arrays section.

[Watch a video tutorial on Mappings.](#)

### ★ Assignment

1. Create a public mapping `[balances]` that associates the key type `[address]` with the value type `[uint]`.
2. Change the functions `[get]` and `[remove]` to work with the mapping `[balances]`.
3. Change the function `[set]` to create a new entry to the `[balances]` mapping, where the key is the address of the parameter and the value is the balance associated with the address of the parameter.

[Check Answer](#)

[Show answer](#)

Next

Well done! No errors.

```
6   mapping(address => uint) public myMap;
7   // Kunal Panjab D20A46ey D20A37
8   mapping(address => uint) public balances;
9
10
11  function get(address _addr) public view returns (uint) { 2095 gas
12    // Mapping always returns a value.
13    // If the value was never set, it will return the default value.
14    return balances[_addr];
15  }
16
17  function set(address _addr) public { 25279 gas
18    // Update the value at this address
19    balances[_addr] = _addr.balance;
20  }
21
22  function remove(address _addr) public { 5588 gas
```

[Explain contract](#)

0 Listen on all transactions



Filter with transaction hash or ad...

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 14

[Watch a video tutorial on Structs.](#)

### ★ Assignment

Create a function `[remove]` that takes a `[uint]` as a parameter and deletes a struct member with the given index in the `[todos]` mapping.

[Check Answer](#)

[Show answer](#)

Next

Well done! No errors.

```
46  }
47  // Kunal Panjab D20A46
48  function remove(uint _index) public { infinite gas
49  | delete todos[_index];
50 }
```

[Explain contract](#)

0 Listen on all transactions



Filter with transaction hash or ad...

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial 15

Another way to update the value is using the `[=]` operator by providing the name of the enum and its member (line 35).

### Removing an enum value

We can use the delete operator to delete the enum value of the variable, which means as for arrays and mappings, to set the default value to 0.

[Watch a video tutorial on Enums.](#)

### ★ Assignment

1. Define an enum type called `[Size]` with the members `S`, `M`, and `L`.
2. Initialize the variable `[size]` of the enum type `[Size]`.
3. Create a getter function `[getSize()]` that returns the value of the variable `[size]`.

[Check Answer](#)

[Show answer](#)

Next

Well done! No errors.

```
15  S
16  M
17  L
18
19  // Default value is the first element listed in
20  // definition of the type, in this case "Pending"
21  Status public status;
22  Size public size;
23  // Returns uint
24  // Pending - 0
25  // Shipped - 1
26  // Accepted - 2
27  // Rejected - 3
28  // Canceled - 4
29  function getSize() public view returns(Size){ 2055 gas
30    return size;
31  }
```

[Explain contract](#)

0 Listen on all transactions



Filter with transaction hash or ad...

AI copilot

Note: The called function should be payable if you send value and the value you send should be less than your current balance. If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Tutorial - 16

**★ Assignment**

- Change the value of the `myStruct` member `foo`, inside the `function f`, to 4.
- Create a new struct `myMemStruct2` with the data location `memory` inside the `function f` and assign it the value of `myStruct`. Change the value of the `myMemStruct2` member `foo` to 1.
- Create a new struct `myMemStruct3` with the data location `memory` inside the `function f` and assign it the value of `myStruct`. Change the value of the `myMemStruct3` member `foo` to 3.
- Let the function `f` return `myStruct`, `myMemStruct2`, and `myMemStruct3`.

Tip: Make sure to create the correct return types for the function `f`.

[Check Answer](#) [Show answer](#)

[Next](#)

Well done! No errors.

```
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

```
function f() {
    uint[] storage _arr;
    mapping(uint => address) storage _map;
    MyStruct storage _myStruct
} internal {
    // do something with storage variables
}

// You can return memory variables
function g(uint[] memory arr) public returns (uint[] memory) {
    // do something with memory array
    arr[0] = 1;
}
//Prajjwal Pandey D20A 37
//Kunal Punjabi D20A 46
calldata _arr) external {
    // do something with calldata array
    // _arr[0] = 1;
}
```

Activate Windows  
Go to Settings to activate Windows...

## Tutorial 17

[Tutorials list](#) [Syllabus](#)

10.1 Transactions - Ether and Wei  
17 / 19

`gwei`  
One `gwei` (giga-wei) is equal to 1,000,000,000 (10<sup>9</sup>) `wei`.  
`ether`  
One `ether` is equal to 1,000,000,000,000,000,000 (10<sup>18</sup>) `wei` (line 11).  
Watch a video tutorial on Ether and Wei.

**★ Assignment**

- Create a `public uint` called `oneWei` and set it to 1 `gwei`.
- Create a `public bool` called `isOneWei` and set it to the result of a comparison operation between 1 `gwei` and 10<sup>9</sup>.

Tip: Look at how this is written for `gwei` and `ether` in the contract.

[Check Answer](#) [Show answer](#)

[Next](#)

Well done! No errors.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 contract EtherUnits {
5     uint public oneWei = 1 wei;
6     // 1 wei is equal to 1
7     bool public isOneWei = 1 wei == 1;
8
9     uint public oneEther = 1 ether;
10    // 1 ether is equal to 10^18 wei
11    bool public isOneEther = 1 ether == 1e18;
12
13    uint public oneGwei = 1 gwei;
14    // 1 ether is equal to 10^9 wei
15    bool public isOneGwei = 1 gwei == 1e9;
16
17
}
// Kunal Punjabi D20A 46
```

## Tutorial 18

The screenshot shows the Remix IDE interface for Tutorial 18. The left panel displays the assignment details for "10.2 Transactions - Gas and Gas Price" with 18/19 steps completed. It includes tips about gas consumption and a video link. The right panel shows the Solidity code for a "Gas" contract:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Gas {
5     uint public i = 0;
6     uint public cost = 170367;
7     // Kunal Punjabi D20A46/ D20A 37
8     // Using up all of the gas that you send causes your transaction to fail.
9     // State changes are undone.
10    // Gas spent are not refunded.
11    function forever() public {
12        // Here we run a loop until all of the gas are spent
13        // and the transaction fails
14        while (true) {
15            i += 1;
16        }
17    }
18 }
```

Below the code, there are "Check Answer" and "Show answer" buttons, and a "Next" button. A green bar at the bottom says "Well done! No errors." On the far right, there are "Activate Windows" and "Go to Settings to activate Windows" links.

## Tutorial 19

The screenshot shows the Remix IDE interface for Tutorial 19. The left panel displays the assignment details for "10.3 Transactions - Sending Ether" with 19/19 steps completed. It includes a tip about testing the contract. The right panel shows the Solidity code for a "Charity" contract:

```
43 }
44
45 function sendViaCall(address payable _to) public payable {
46     // Call returns a boolean value indicating success or failure.
47     // This is the current recommended method to use.
48     (bool sent, bytes memory data) = _to.call{value: msg.value}("");
49     require(sent, "Failed to send Ether");
50 }
51 //Pratik Pandey D20A 37
52 //Kunal Punjabi D20A46
53 address public owner;
54
55 constructor() {
56     owner = msg.sender;
57 }
58
59 function donate() public payable {
60     // 165452 gas 141000 gas
61 }
62
63 function withdraw() public {
64     uint amount = address(this).balance;
65     (bool sent, bytes memory data) = owner.call{value: amount}("");
66     require(sent, "Failed to send Ether");
67 }
68 }
```

Below the code, there are "Check Answer" and "Show answer" buttons, and a "Next" button. A green bar at the bottom says "Well done! No errors." On the far right, there are "Activate Windows" and "Go to Settings to activate Windows" links.

## Conclusion :

Through this experiment, the fundamentals of Solidity programming were explored by completing practical assignments in the Remix IDE. Concepts such as data types, variables, functions, visibility, modifiers, constructors, control flow, data structures, and transactions were implemented and understood. The hands-on practice helped in designing, compiling, and deploying smart contracts on the Remix VM, thereby strengthening the understanding of blockchain concepts. This experiment provided a strong foundation for developing and managing smart contracts efficiently.